



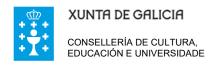




Busca en Internet información sobre estas cuestiones:

 Modifica el ejemplo del fichero Celsius_Fahrenheit (número de capas, por ejemplo, número de valores de prueba, etc.). Muestra el código final y ¿el sistema sigue siendo correcto en sus valores?

```
# RED NEURONAL CELSIUS - FAHRENHEIT
# Importación de Framewordks
import tensorflow as tf
import numpy as np
import random
celsius = np.array( [] )
for i in range( random.randint( 10, 50 ) ):
  grado = random.randrange( -1000, 1000, 1 )
  error = (.5 - random.random()) * grado
  celsius = np.append( celsius, grado ) + error
fahrenheit = np.array( [] )
for grado celsius in celsius:
```

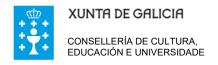








```
grado_fahrenheit = grado_celsius * 1.8 + 32
  error = (.5 - random.random()) * grado
  fahrenheit = np.append( fahrenheit, grado fahrenheit +
error )
   tf.keras.layers.Dense
conexiones son todas
   input shape = [ 1 ]
        Define el modelo de capa de tipo Secuencial con una
capa = tf.keras.layers.Dense( units = 4, input_shape = [ 1 ]
modelo = tf.keras.Sequential( [ capa ] )
```









```
modelo.compile(
    optimizer = tf.keras.optimizers.Adam( 0.05 ), # Nivel de
    loss = 'mean squared error'
# Entrenamiento del modelo
                                          'celsius'
   - Datos de salida esperados:
   - Número de interaciones:
print( "Comenzamos el entrenamiento ..." )
historial = modelo.fit( celsius,
                        fahrenheit,
                        epochs = 500,
                        verbose = False )
print( "Modelo ya entrenado" )
# Muestra de los resultados obtenidos
import matplotlib.pyplot as plt
plt.xlabel( "Celsius" )
plt.ylabel( "Fahrenheit" )
plt.scatter( celsius, fahrenheit, color = 'green' )
```









```
• plt.plot( celsius, modelo.predict( celsius ) )
• plt.show()
• 
• print( "Variables internas del modelo" )
• print( capa.get_weights() )
• 
• print( "Hagamos una predicción" )
• #celsius_prueba = random.randint( -500, 500 )
• celsius_prueba = 100
• resultado = modelo.predict( [ celsius_prueba ] )
• print( str( celsius_prueba ) + "ºC son " + str( resultado ) + "º fahrenheit" )
```

Al modificar el numero de units en una capa densa de una red neuronal, estoy aumentando la capacidad del modelo para aprender representaciones mas complejas de los datos, aprender patrones complejos pero tambien lo hace mas propenso al sobreajuste y aumenta el tiempo de entrenamiento.









 Modifica el ejemplo del Árbol_de_decisión. Por ejemplo, busca cómo modificar el número de capas y ramas. Muestra el código final y ¿el sistema sigue siendo correcto en sus valores?

```
INSTALAMOS LIBRERÍAS
import pandas as pd
import matplotlib.pyplot as plt
# CARGAMOS LOS DATOS
df = pd.read_csv( "problemas_del_corazon.csv" )
   DATOS DE ENTRENAMIENTO Y PRUEBA
from sklearn.model selection import train test split
datos = [ "edad", "genero", "presion" ]
objetivo = [ "uso de internet" ]
datos_entrena, datos_prueba, clase_entrena, clase_prueba =
train_test_split(
    df[ datos
    df[ "diabetico"
```









```
test_size = 0.25
    CREACIÓN DEL ÁRBOL DE DECISIÓN
from sklearn import tree
arbol_decision = tree.DecisionTreeClassifier(
    criterion = "gini",  # gini, entropy, log_loss
splitter = "best",  # best, random
    max_depth = 4
arbol = arbol_decision.fit(datos_entrena, clase_entrena)
plt.figure( figsize = ( 20, 15 ) )
tree.plot_tree( arbol,
                max_depth = 4,
                feature_names = datos,
                filled = True,
                impurity = False,
                precision = 0
plt.show()
```

Modifique la profundidad maxima del arbol a 4 y el criterion a gini, esto hace que se modifique la forma de evaluar la calidad de las divisiones del arbol, reducir la profundidad maxima del arbol hace que el modelo sea menos complejo pero tambien reduce su capacidad de encontrar patrones complejos en los datos, dando lugar a un posible mayor sesgo. Pero aun con todo los datos siguen siendo validos, pero el rendimiento del modelo puede verse afectado por los cambios.